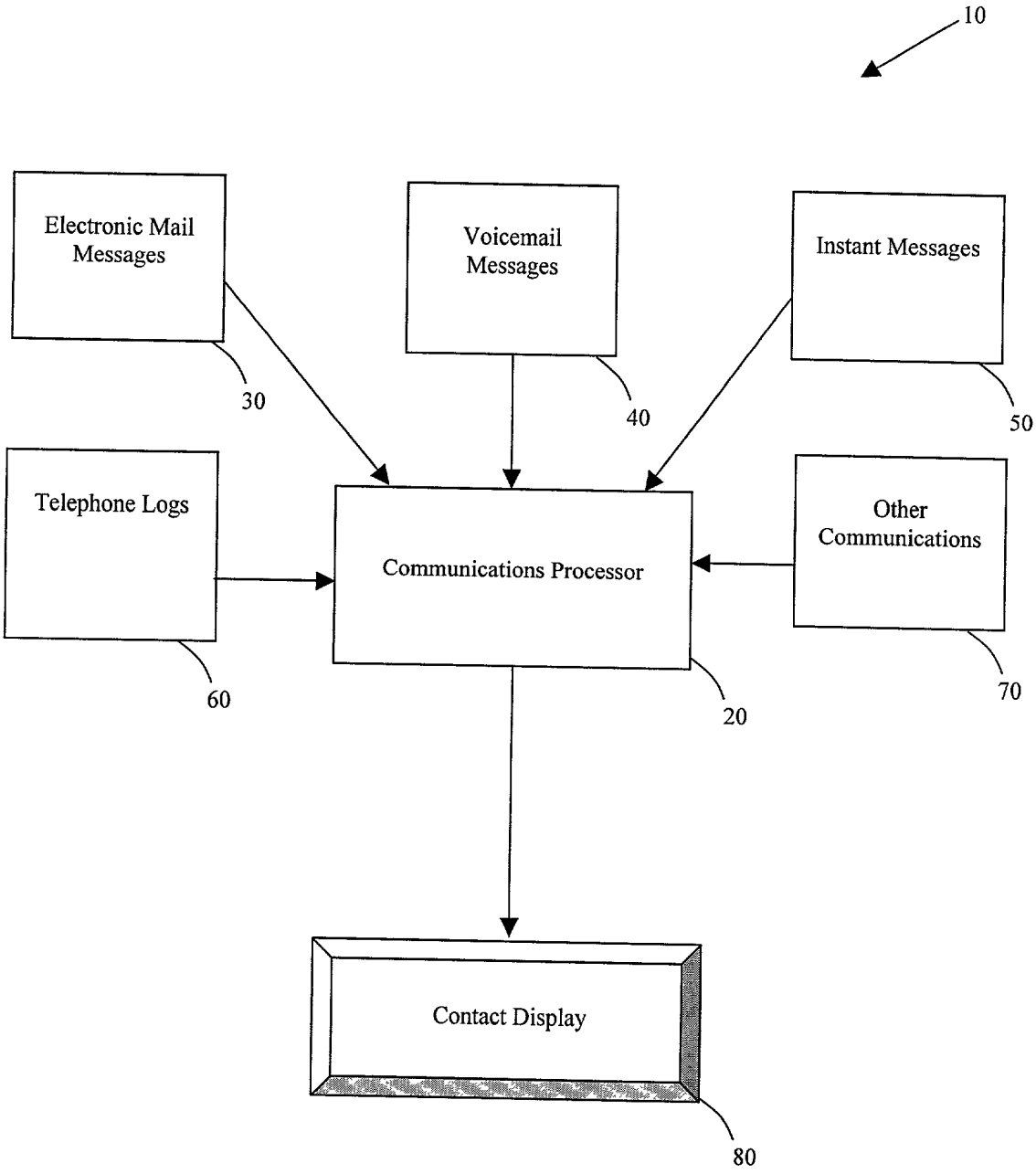
10
Electronic Mail Messages
30
Voicemail Messages
40
Instant Messages
50
Telephone Logs
60
Communications Processor
20
Other Communications
70
Contact Display
80

FIG. 1

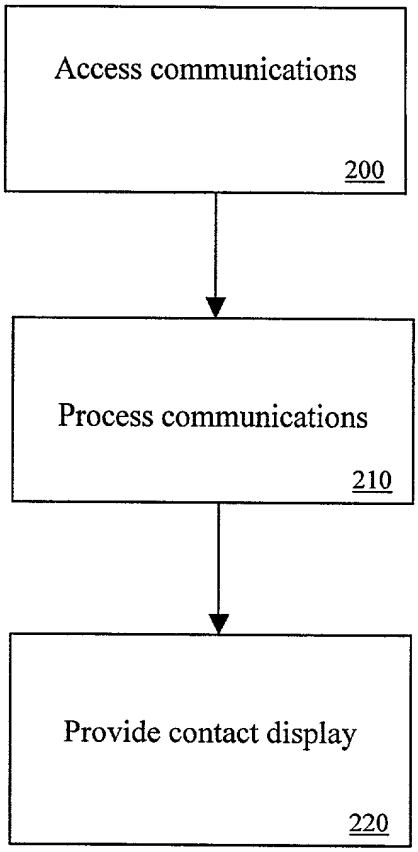
Access communications
200
Process communications
210
Provide contact display
220

FIG. 2

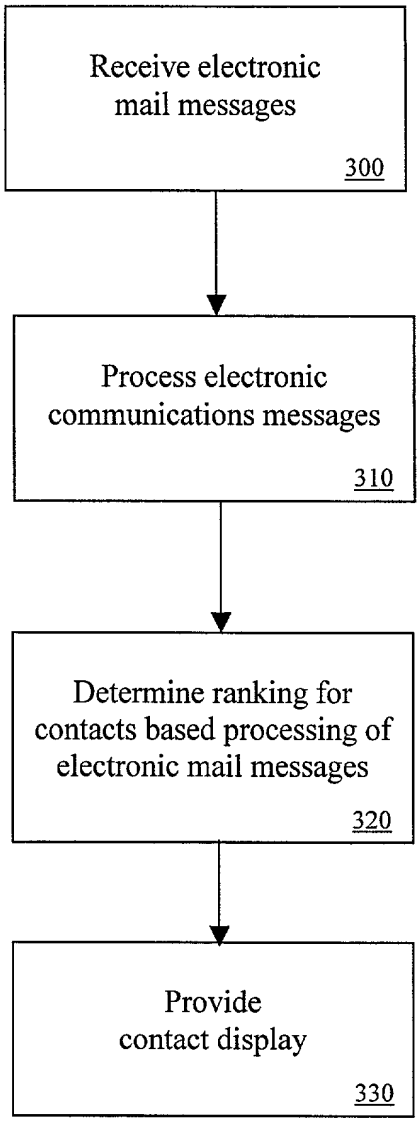
Receive electronic mail messages
300
Process electronic communications messages
310
Determine ranking for contacts based processing of electronic mail messages
320
Provide contact display
330

FIG. 3

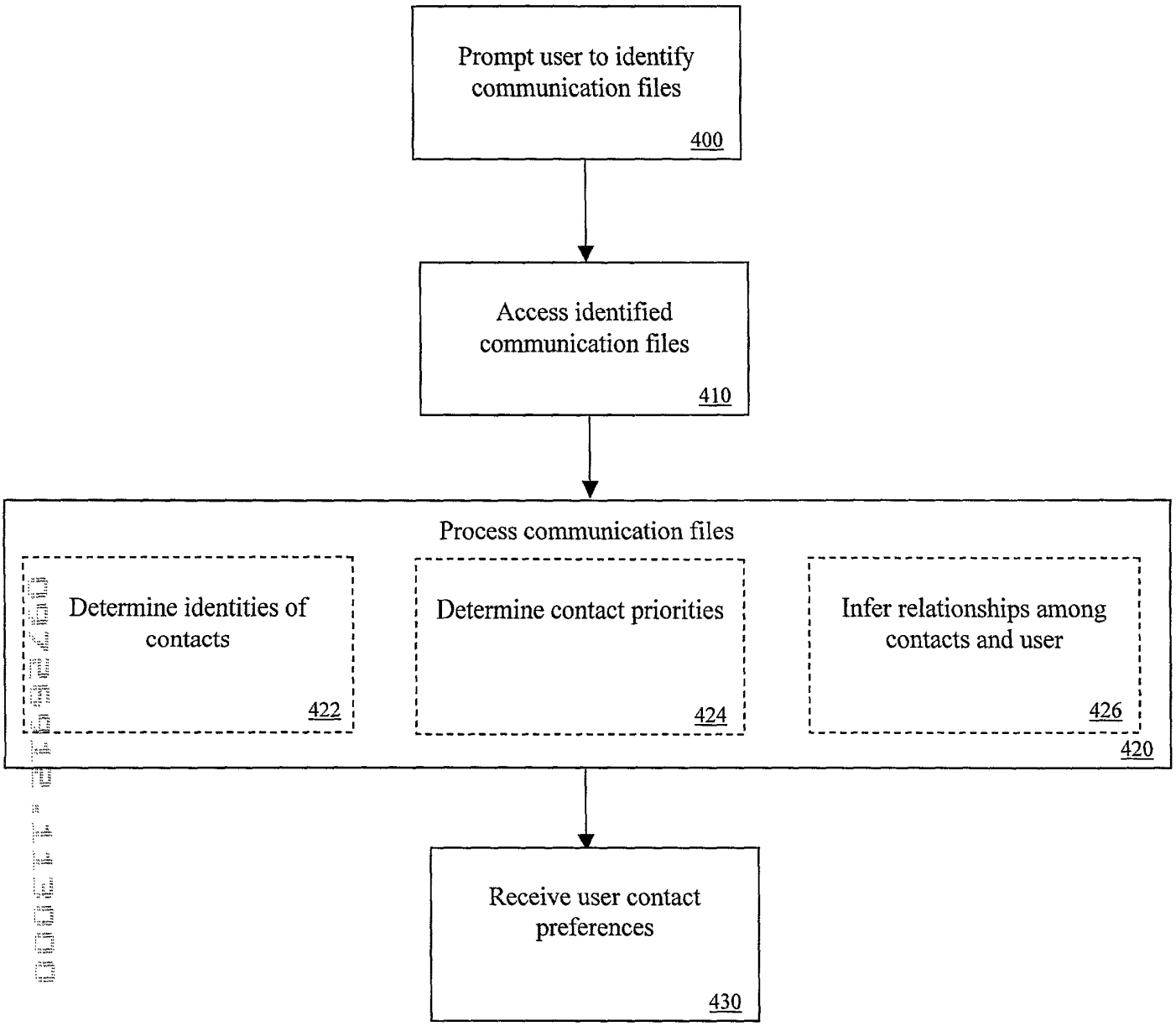
Prompt user to identify communication files
400
Access identified communication files
410
Process communication files
Determine identities of contacts
422
Determine contact priorities
424
Infer relationships among contacts and user
426
420
Receive user contact preferences
430

FIG. 4

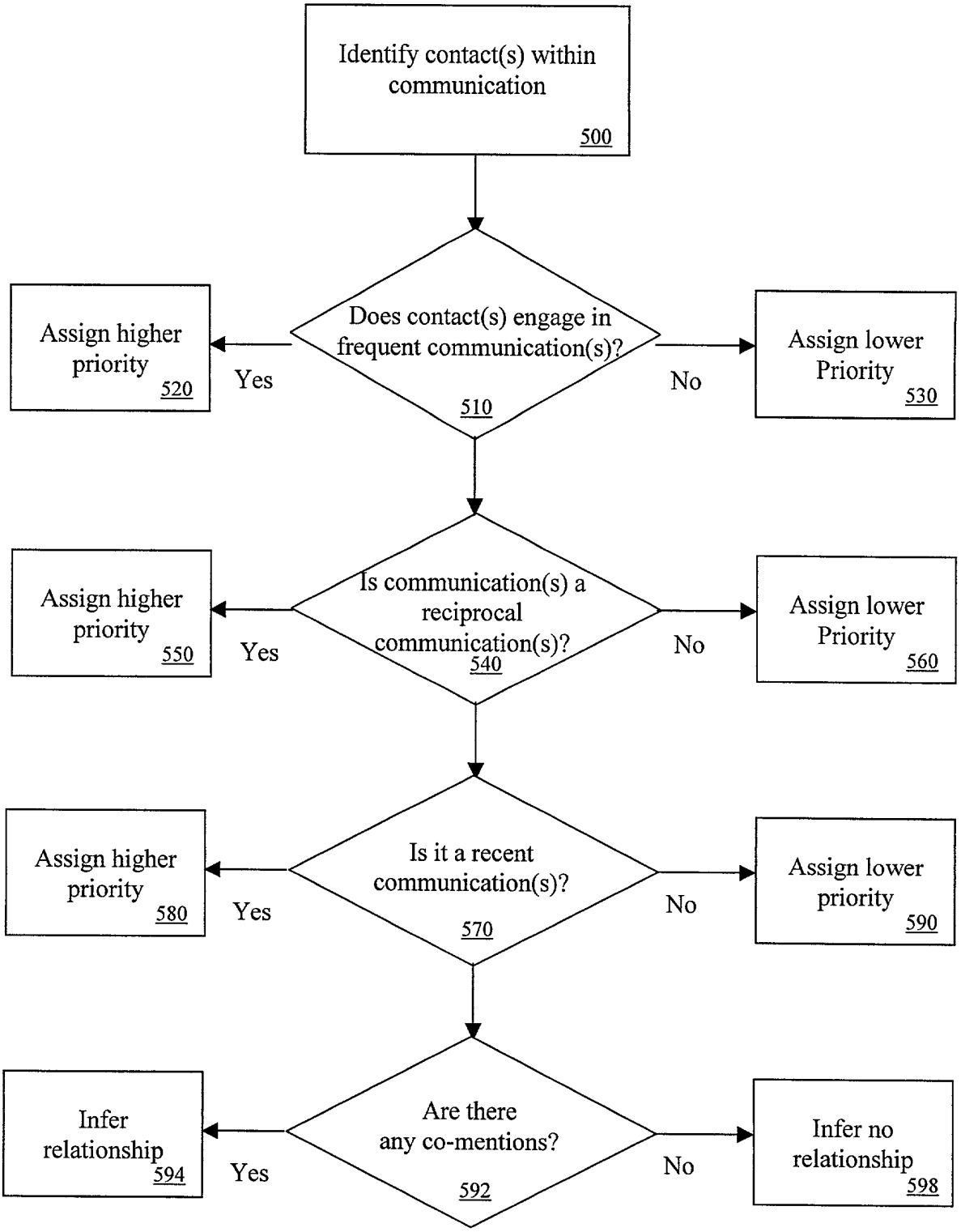
Identify contact(s) within communication
500
Does contact(s) engage in frequent communication(s)?
510
Yes
No
Assign higher priority
520
Assign lower Priority
530
Is communication(s) a reciprocal communication(s)?
540
Yes
No
Assign higher priority
550
Assign lower Priority
560
Is it a recent communication(s)?
570
Yes
No
Assign higher priority
580
Assign lower priority
590
Are there any co-mentions?
592
Yes
No
Infer relationship
594
Infer no relationship
598

FIG. 5

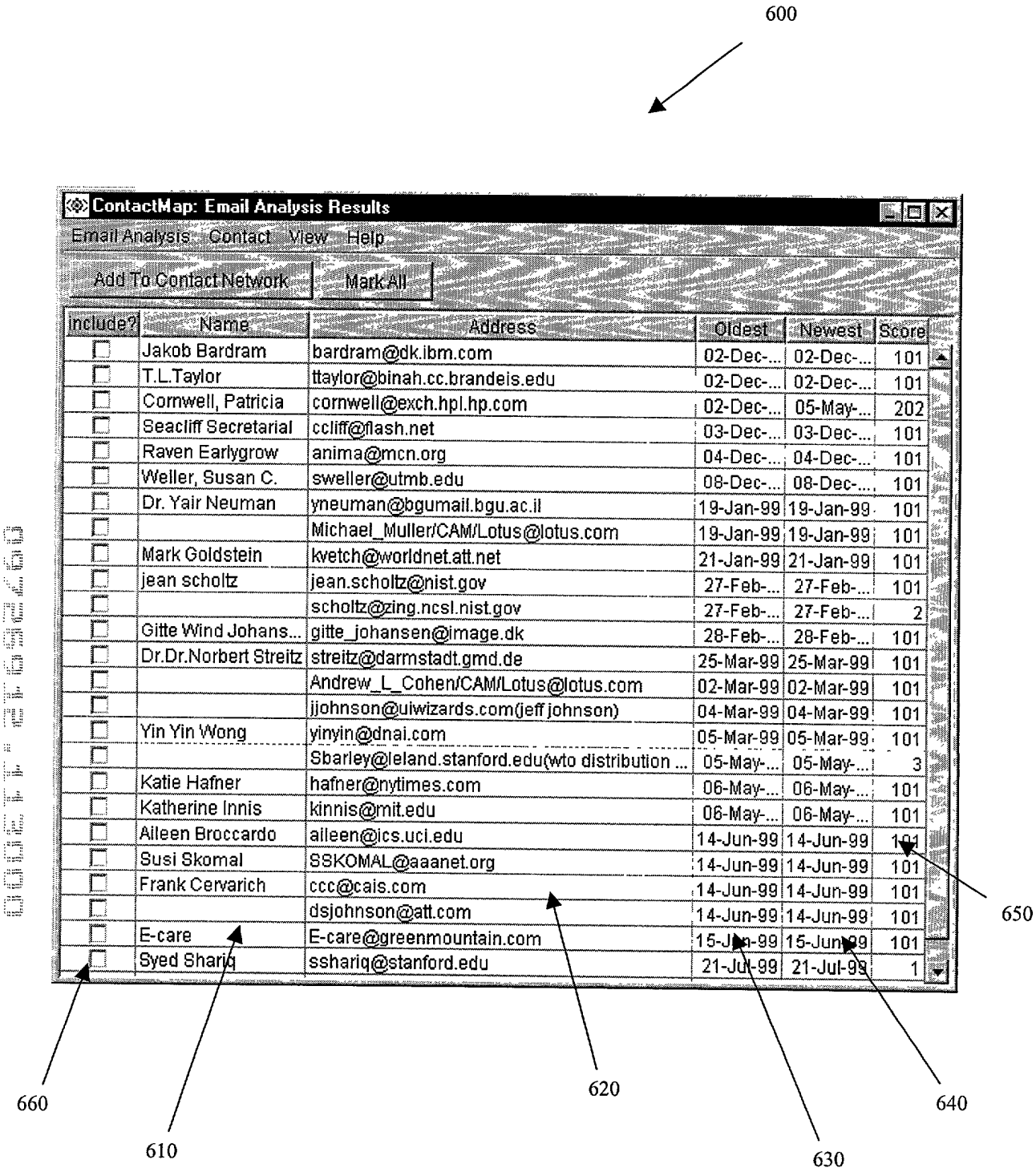
600
ContactMap: Email Analysis Results
Email Analysis Contact View Help
Add To Contact Network
Mark All
Include? | Name | Address | Oldest | Newest | Score
Jakob Bardram | bardram@dk.ibm.com | 02-Dec-... | 02-Dec-... | 101
T.L.Taylor | ttaylor@binah.cc.brandeis.edu | 02-Dec-... | 02-Dec-... | 101
Cornwell, Patricia | cornwell@exch.hpl.hp.com | 02-Dec-... | 05-May-... | 202
Seacliff Secretarial | ccliff@flash.net | 03-Dec-... | 03-Dec-... | 101
Raven Earlygrow | anima@mcn.org | 04-Dec-... | 04-Dec-... | 101
Weller, Susan C. | sweller@utmb.edu | 08-Dec-... | 08-Dec-... | 101
Dr. Yair Neuman | yneuman@bgumail.bgu.ac.il | 19-Jan-99 | 19-Jan-99 | 101
Michael_Muller/CAM/Lotus@lotus.com | 19-Jan-99 | 19-Jan-99 | 101
Mark Goldstein | kvetch@worldnet.att.net | 21-Jan-99 | 21-Jan-99 | 101
jean scholtz | jean.scholtz@nist.gov | 27-Feb-... | 27-Feb-... | 101
scholtz@zing.ncsl.nist.gov | 27-Feb-... | 27-Feb-... | 2
Gitte Wind Johans... | gitte_johansen@image.dk | 28-Feb-... | 28-Feb-... | 101
Dr.Dr.Norbert Streitz | streitz@darmstadt.gmd.de | 25-Mar-99 | 25-Mar-99 | 101
Andrew_L_Cohen/CAM/Lotus@lotus.com | 02-Mar-99 | 02-Mar-99 | 101
jjohnson@uiwizards.com(jeff johnson) | 04-Mar-99 | 04-Mar-99 | 101
Yin Yin Wong | yinyin@dnai.com | 05-Mar-99 | 05-Mar-99 | 101
Sbarley@leland.stanford.edu(wto distribution ... | 05-May-... | 05-May-... | 3
Katie Hafner | hafner@nytimes.com | 06-May-... | 06-May-... | 101
Katherine Innis | kinnis@mit.edu | 06-May-... | 06-May-... | 101
Aileen Broccardo | aileen@ics.uci.edu | 14-Jun-99 | 14-Jun-99 | 101
Susi Skomal | SSKOMAL@aaanet.org | 14-Jun-99 | 14-Jun-99 | 101
Frank Cervarich | ccc@cais.com | 14-Jun-99 | 14-Jun-99 | 101
dsjohnson@att.com | 14-Jun-99 | 14-Jun-99 | 101
E-care | E-care@greenmountain.com | 15-Jun-99 | 15-Jun-99 | 101
Syed Shariq | sshariq@stanford.edu | 21-Jul-99 | 21-Jul-99 | 1
650
660
610
620
630
640

FIG. 6

```
/* -*-Java-*-
********************************************************************************
*
* File:         MATContactCounts.java
* RCS:          $Header: $
* Description:
* Author:       John Hainsworth
* Created:      Fri Sep 03 15:17:42 1999
* Modified:     Thu Sep 09 16:15:41 1999 (Michael L. Creech) mike@home
* Language:     Java
* Package:      com.att.research.mat.email
* Status:       Experimental (Do Not Distribute)
*
* (c) Copyright 1999, AT&T, all rights reserved.
*
********************************************************************************
*
* Revisions:
*
* Thu Sep 09 16:15:31 1999 (Michael L. Creech) mike@home
*  Added getOldest(), and getNewest().
* Fri Sep 03 15:17:46 1999 (Michael L. Creech) mike@home
*  Made getOrderingValue() public.
********************************************************************************
*/

package com.att.research.mat.email;

import java.io.PrintWriter;
import java.io.IOException;
import java.util.Date;

/** Statistics about mail to and from one e-mail address.
 * These statistics are all redundant with respect to the data,
 * so none of them are saved with the data.
 */

public class MATContactCounts {
  /** Copyright (c) 1999 AT&T Labs */
  static public String Copyright = "Copyright (c) 1999 AT&T Labs";

  static final private int CountNONE = -1;
  static final private double DCountNONE = -1.0;

  static final int FromME = 0;
  static final int FromOTHER = FromME + 1;
  static private final int From_N = FromOTHER + 1;
  static private final String StrFrom[] = { "My", "" };

  /** seen in an original message */
  static final int KindCOMPOSITION = 0;
  /** seen in a reply message */
  static final int KindREPLY = KindCOMPOSITION + From_N;
  /** seen in a forwarded message */
  static final int KindFORWARD = KindREPLY + From_N;
  static private final int Kind_N = KindFORWARD + From_N;
  static private final String StrKind[] = { "", "Repl", "Fwd" };

  /** seen in a "To:" header */
  static final int WhereTO = 0;
  /** seen in a "From:" header */
  static final int WhereFROM = WhereTO + Kind_N;
  /** seen in a "Cc:" header */
  static final int WhereCC = WhereFROM + Kind_N;
  /** seen in a "Bcc:" header */
  static final int WhereBCC = WhereCC + Kind_N;
  static private final int Where_N = WhereBCC + Kind_N;
  static private final String StrWhere[] = { "To", "From", "Cc", "Bcc" };
```

FIG. 7a

```
// Java will initialize these to 0
private int counts[] = new int[Where_N];

Date newest;
Date oldest;

public Date getOldest ()
  {
    return oldest;
  }

public Date getNewest ()
  {
    return newest;
  }

/** @param what Should always be of the form (From?? + Kind?? + Where??) */
void incrementCount(Date d, int what) {
  counts[what]++;
  if (null != d) {
    if (null == newest) {
      oldest = newest = d;
    } else {
      long nms = d.getTime();
      if (nms > newest.getTime()) {
        newest = d;
      } else if (nms < oldest.getTime()) {
        oldest = d;
      }
    }
  }
}

/** @param what Should always be of the form (From?? + Kind?? + Where??) */
int getCount(int what) {
  return counts[what];
}

/** Set all counters to zero. */
void clear() {
  int i = 0;
  while (i < Where_N) {
    counts[i++] = 0;
  }
  nonZeroTotal = CountNONE;
  orderingValue = CountNONE;
  oldest = newest = null;
  duration = CountNONE;
  density = CountNONE;
  freshness = DCountNONE;
}

/** Get total number of messages this contact actually appeared in.
 * Since we add To:s to Cc:s and Cc:s to Bcc:s, we just need to count
 * From:s and Bcc:s. */
private int getMessageCount() {
  return (counts[FromME + KindCOMPOSITION + WhereFROM] +
          counts[FromME + KindREPLY + WhereFROM] +
          counts[FromME + KindFORWARD + WhereFROM] +
          counts[FromOTHER + KindCOMPOSITION + WhereFROM] +
          counts[FromOTHER + KindREPLY + WhereFROM] +
          counts[FromOTHER + KindFORWARD + WhereFROM] +
          counts[FromME + KindCOMPOSITION + WhereBCC] +
          counts[FromME + KindREPLY + WhereBCC] +
          counts[FromME + KindFORWARD + WhereBCC] +
          counts[FromOTHER + KindCOMPOSITION + WhereBCC] +
          counts[FromOTHER + KindREPLY + WhereBCC] +
          counts[FromOTHER + KindFORWARD + WhereBCC]);
}
```

FIG. 7b

```
private long duration = CountNONE;
public long getDuration() {
  if (CountNONE == duration) {
    if (null != oldest) {
      duration = newest.getTime() - oldest.getTime();
    } else {
      duration = 0;
    }
  }
  return duration;
}

private long density = CountNONE;
public long getDensity() {
  if (CountNONE == density) {
    long dur = getDuration();
    if (0 < dur) {
      density = getMessageCount() / dur;
    } else {
      density = 0;
    }
  }
  return density;
}

private double freshness = DCountNONE;
public double getFreshness() {
  if (DCountNONE == freshness) {
    freshness = (((double)getDensity()) /
                 ((double)((new Date()).getTime() - newest.getTime())));
  }
  return freshness;
}

private int nonZeroTotal = CountNONE;
int getNonZeroTotal() {
  if (CountNONE == nonZeroTotal) {
    nonZeroTotal = 0;
    int i = 0;
    while (i < Where_N) {
      nonZeroTotal += counts[i];
      i += 1;
    }
  }
  return nonZeroTotal;
}

private int orderingValue = CountNONE;

// MLC:
public int getOrderingValue() {
  if (CountNONE == orderingValue) {
    orderingValue =
      counts[FromME + KindREPLY + WhereTO] * 1000000 +
      counts[FromME + KindCOMPOSITION + WhereTO] * 10000 +
      counts[FromOTHER + KindREPLY + WhereFROM] * 100 +
      counts[FromOTHER + KindCOMPOSITION + WhereFROM] * 100 +
      getNonZeroTotal();
  }
  return orderingValue;
}
```

FIG. 7c

```
  private String print1(String leader, PrintWriter ps, int what)
    throws IOException {
    if (0 != counts[what]) {
      int iFrom = what % From_N;
      int iKind = (what % Kind_N) / From_N;
      int iWhere = what / Kind_N;
      ps.print(leader+StrFrom[iFrom]+StrKind[iKind]+StrWhere[iWhere]+
               "="+counts[what]);
      leader = " ";
    }
    return leader;
  }

  public void print(PrintWriter ps, int format) throws IOException {
    switch (format) {
    case MATData.FormatANALYSIS:
      String leader = " {";
      int what = 0;
      leader = print1(leader, ps, FromME + KindREPLY + WhereTO);
      leader = print1(leader, ps, FromME + KindCOMPOSITION + WhereTO);
      leader = print1(leader, ps, FromOTHER + KindREPLY + WhereFROM);
      leader = print1(leader, ps, FromOTHER + KindCOMPOSITION + WhereFROM);
      while (what < Where_N) {
        switch (what) {
        case FromME + KindREPLY + WhereTO:
        case FromME + KindCOMPOSITION + WhereTO:
        case FromOTHER + KindREPLY + WhereFROM:
        case FromOTHER + KindCOMPOSITION + WhereFROM:
          break;
        default:
          leader = print1(leader, ps, what);
          break;
        }
        what++;
      }
      if (leader.equals(" ")) {
        ps.print("}");
      }
      break;
    case MATData.FormatTERMINAL:
      int what2 = 0;
      String leader2 = " {";
      while (what2 < Where_N) {
        leader = print1(leader2, ps, what2++);
      }
      if (leader2.equals(" ")) {
        ps.print("}");
      }
      ps.print(" score="+getOrderingValue());
      break;
    case MATData.FormatDATABASE:
      break;
    }
  }
}
```

FIG. 7d

```
package com.att.research.mat.email;

import com.att.research.mat.utility.UniqueIDs;
import com.att.research.mat.utility.NumberedMember;
import java.util.Enumeration;
import java.io.PrintWriter;
import java.io.IOException;
import java.io.StringWriter;
import java.util.Date;

/** A list of contacts that appeared together on a To, Cc, or Bcc line. */

public class MATContactGroup extends UniqueIDs {
  /** Copyright (c) 1999 AT&T Labs */
  static public String Copyright = "Copyright (c) 1999 AT&T Labs";

  /** @serial cc not stored */
  private MATContacts cc;

  // MLC:
  public MATContacts getMATContacts ()
    {
      return cc;
    }

  public MATContactGroup(MATContacts cc) {
    this.cc = cc;
  }

  public Enumeration elements() { return elements(cc); }

  public void hideUnimportantMembers(int threshold) {
    Enumeration e = elements(cc);
    while (e.hasMoreElements()) {
      ((MATContact)e.nextElement()).hideIfUnimportant(threshold);
    }
  }

  public boolean containsMe() {
    Enumeration e = elements(cc);
    while (e.hasMoreElements()) {
      if (((MATContact)e.nextElement()).isMe()) {
        return true;
      }
    }
    return false;
  }

  /** @return Unique id of removed member. */
  public int removeMe() {
    int i = 0;
    Enumeration e = elements(cc);
    while (e.hasMoreElements()) {
      MATContact c = (MATContact)e.nextElement();
      if (c.isMe()) {
        removeElementAt(i);
        return c.getUniqueID();
      }
      i++;
    }
    return NumberedMember.IdNONE;
  }
```

FIG. 8a

```
public MATContactGroup updateToRemoveMe() {
      int removedMe = removeMe();
      if (NumberedMember.IdNONE != removedMe) {
        int others = toMATContactID();
        MATContactGroup meGroup = new MATContactGroup(cc);
        meGroup.addElement(removedMe);
        meGroup.addElement(others);
        meGroup.sortIds();
        return meGroup;
      } else {
        return null;
      }
  }

  public int toMATContactID() {
    switch (size()) {
    case 0:
      return NumberedMember.IdNONE;
    case 1:
      return uniqueIdAt(0);
    default:
      int removedMe = removeMe();
      if (NumberedMember.IdNONE != removedMe) {
        int others = toMATContactID();
        MATContactGroup meGroup = new MATContactGroup(cc);
        meGroup.addElement(removedMe);
        meGroup.addElement(others);
        meGroup.sortIds();
        MATContact c = cc.find(meGroup);
        c.setHidden(true);
        return c.getUniqueID();
      } else {
        sortIds();
        return cc.find(this).getUniqueID();
      }
    }
  }

  /** Calls {@link com.att.research.mat.email.MATContact#incrementCount(Date,int)}
   * for each member. */
  void incrementCount(Date d, int what) {
    Enumeration e = elements(cc);
    while (e.hasMoreElements()) {
      ((MATContact)e.nextElement()).incrementCount(d, what);
    }
  }


  /** Needed by the contact the viewer. */
  // This method MUST use cc, and cannot get it as an argument.
  // This is the only true reason that cc must be a member of this object.
  public String toString() {
    StringWriter sw = new StringWriter();
    PrintWriter ps = new PrintWriter(sw);
    try {
      print(ps, cc, " ", MATData.FormatADDRESS_ONLY);
    } catch (IOException e) {
    }
    return sw.toString();
  }
}
```

FIG. 8b